Advanced monitoring and smart auto-scaling of NoSQL systems

Arnaud Schoonjans, Bert Lagaisse, Wouter Joosen
iMinds-DistriNet, KU Leuven
3001 Leuven, Belgium
firstname.lastname@cs.kuleuven.be

ABSTRACT

Recent years have shown that RDBMS systems do not always meet the performance and scalability requirements of today's applications. Horizontal scalability is hindered by the ACID properties and the normalized data model these systems use. For this reason, a whole new range of database systems (NoSQL systems) has emerged. This paper focusses on eventual consistent storage systems, which have a certain inconsistency window after an update. Within this window different replicas contain a different version of a certain data item.

While RDBMS systems provide strong transactional semantics, this is not the case for eventual consistent storage systems. The level of consistency is often configurable, but figuring out the optimal configuration is not a trivial task. Next to that, recent research has shown that the size of the inconsistency window can change over time, considering a fixed configuration.

In this Phd research we envision a solution where all consistency-related parameters are managed by an SLA-driven autonomous system. Continuously monitoring the size of the inconsistency window allows dynamic reconfiguration and re-provisioning of the database cluster to keep the inconsistency window under a certain limit. As such, more guarantees can be provided to the application programmer.

Categories and Subject Descriptors

C.2 [**COMPUTER-COMMUNICATION NETWORKS**]; C.2.4 [**Distributed Systems**]; H.2 [**DATABASE MANAGEMENT**]; H.2.7 [**Database Administration**]

Keywords

NoSQL, inconsistency window

Middleware Doctoral Symposium '15, December 07-11, 2015, Vancouver, BC, Canada

DOI: http://dx.doi.org/10.1145/2843966.2843969

1. INTRODUCTION

Relational database systems (RDBMS systems) have been considered as a one-size-fits-all-solution for a long time. These systems have clear transactional properties, known as the ACID (atomicity, consistency, isolation and durability) properties. All these properties together provide a very convenient environment to the application programmer, since he or she can rely on them while writing applications.

Recent years have shown that RDBMS systems do not always succeed in meeting today's scalability requirements. Large web-application, for example, have to serve a large amount of users at (i) low latencies, since web-applications are most of all interactive applications and (ii) at high availability, since users expect the web-application to be available all the time. The vertical scaling technique, used by RDBMS systems, is very disadvantageous because it requires expensive hardware and only provides limited scaling. The ACID properties, prevent RDBMS systems from scaling horizontally, since distributing the data across multiple nodes would require distributed transactions to satisfy the consistency requirement of the ACID properties. Distributed transaction are very slow and infeasible for operations that require low latencies [2, 7]. As applications pop-up with more and more users, horizontal scalability becomes mandatory to meet the required performance.

To meet the aforementioned requirements, several new categories of database systems have emerged, called NoSQL (Not only SQL) storage systems. These system apply a denormalized data model together with an appropriate sharding and replication scheme to increase performance and availability. As such, the database system can scale out to thousands of nodes.

Of course, all these benefits come at a certain cost given the CAP-theorem [5, 9] of E. Brewer. This theorem says that a distributed storage system is only able to support one of the following three properties at the same time:

- Consistency: The database system has a single, up-to-date copy of the data;
- Availability: The storage system will remain available (for updates);
- Partition tolerance: The storage system will continue to operate under network partitions.

Since network partitions cannot be avoided in practice, a storage system has to make a trade-off between consistency and availability. It has to be mentioned that this trade-off is not a black-white decision. It is for example possible to

have a strong consistent storage system under normal operations (non-failure conditions), while the system becomes eventually consistent when node and/or network failure occurs. This gray scale leads to a large, heterogeneous group of NoSQL systems that make different architectural decisions regarding: sharding, replication, consistency guarantees, transaction support, ability to overcome failure, data model (key-value store, document store, column-store, ...), ability to scale, etc.

Most large applications need high availability and high performance. As such, they prefer high availability above strong consistency. The result is a group of NoSQL storage systems which only offer eventual consistency instead of strong consistency. This means that there exists a certain time window after an update where both the new as well as old version of the updated data items are observable by clients. This time window is called the inconsistency window of the eventual consistent storage system.

2. PROBLEM STATEMENT

Since eventual consistent storage systems only guarantee to become consistent after an undefined amount of time, it is substantially more difficult to write an application against eventually consistent storage systems. The application programmer has weaker guarantees about the consistency properties of the database system, while strong guarantees are provided by RDBMS systems. Is the inconsistency window in the order of milliseconds, seconds, minutes,...? Appropriate configuration[1] is required to keep the inconsistency window under a certain limit, but finding the right configuration is not an easy task either.

Several research papers and industry blog articles show that good knowledge about the size of the inconsistency window is important for the correct working of the application. Large companies, such as Netflix[6], have been doing extensive testing to determine the actual consistency guarantees of their platform. Next to that, there has been significant interest from academia. Several research papers describe in detail what eventual consistency is [1, 11, 14] and provide different perspective to look at it. Different database technologies have been subject to consistency benchmarks that try to give better insight into the the size of the inconsistency window in practice [13, 3, 10, 15]. The benchmark results show that the size of the inconsistency window highly depends on the database technology and the specific configuration.

Bermach et. al [4] have shown that the size of the inconsistency window can change over time. This is probably due to the fact that the cloud infrastructure is a shared resource. Many different customers allocate and release resources according to their needs, which causes load changes on the cloud infrastructure and as a consequence changes in the size of the inconsistency window. The same can happen in non-virtualized environments when the resources of the physical nodes are shared with other services. As a consequence, the size of the inconsistency window not only depends on the specific database technology and its configuration, but also on more dynamic parameters like the load on the database and platform that database system is running on.

[1]Examples of these consistency-related configuration parameters include: read/write consistency level in Cassandra[8], read preference and write concern in MongoDB[12], etc.

To conclude, the consistency-related configuration parameters of a certain storage system and application can only be determined in an optimal way at run-time depending on for example, the load on the cloud infrastructure, the amount of nodes in the database cluster, etc. More dynamic configuration management is required to choose the right configuration parameters for a certain application.

3. MOTIVATION

Controlling the size of the inconsistency window is important for several reasons. First of all, a drift in the size of the window can cause bad user experience and serious money loss to the owner of the application. In case of an e-commerce application, changes are considerably larger to have a double booking when the inconsistency window gets bigger. An optimal trade-off is required between compensation cost due to database inconsistencies and the financial cost and the performance overhead of stronger consistency requirements.

Second, configuring the consistency-related parameters in a static way can lead to a too strict configuration regarding consistency. Since, performance and consistency are inversely correlated, this causes an overallocation of resources to meet the required performance. Dynamic management of the consistency-related parameters saves money due to a better usage of the pay-as-you-use billing model in the cloud.

4. AIMS AND OBJECTIVES

The objective is to develop a service level agreement (SLA) driven autonomous system to automatically change the configuration and deployment of an eventual consistent system regarding the requirements written down in the SLA of the application. The SLA we talk about is an extended version of traditional SLAs, since it not only defines constraints on performance and availability, but also on the maximum size of the inconsistency window. Since the configuration of non-functional requirements such as consistency, availability and performance influence each other, autonomously choosing the right configuration will help in managing the database system as a whole.

The goal is not to develop a completely new database system, but rather to build this functionality on top of existing database systems, such as Cassandra, MongoDB, CouchDB, Riak, etc., widely applied in big applications. As such, applications can benefit from the specific trade-offs on performance, availability and scalability embedded in the architectural decisions of these systems, while acquiring more specific guarantees regarding consistency.

The autonomous system will release the system administrator and the application programming from having to guess consistency-related parameters which do not have a predicable impact on the overall application. As such they should be derived from the non-functional requirements of the application defined within the SLA.

5. RESEARCH QUESTIONS

The following research questions are subject of the Phd research:

Is is possible to measure the size of the inconsistency window in an efficient way?
This is an important criteria for building the autonomous

system. If the performance of monitoring the size of the inconsistency window is larger than the cost of overallocating resources to keep the consistency window low, monitoring is not useful at all. Several approximation techniques might give an estimate on the actual inconsistency window. For example, monitoring round trip times to the database, performing a read-after-write operation on a dummy table in the database to determine consistency artificially, etc. The evaluation of this question has to take into account both the financial and performance cost of the monitoring activities. This involves the cost of additional latency for production queries when they are subject to consistency measurements, the cost of additional load on the database due to artificial queries to measure consistency, the cost of the computing power required to process and analyse these consistency measurements, etc.

To which extent is it possible to derive consistency-related parameters from an SLA that puts restrictions on performance, consistency and availability?
It is not clear whether enough information can be extracted from the database system to make clear decisions regarding the consistency-related configuration parameters. The appropriate configuration and the impact of reconfiguration decisions will be specific to a certain database technology and might not be easy to determine.

What is the overhead of possible reconfiguration actions on the inconsistency window and the overall performance of the database system?
Several reconfiguration actions are possible to decrease the size of the inconsistency window, eg., changing the consistency levels of the query operations, changing the replication factor, increasing the amount of nodes, etc. All these reconfiguration operations will have an impact on the performance as well as the consistency of the database system. We see two important items the autonomous reconfiguration has to take care of.
First of all, it is important that the decisions made by the autonomous system converge to a steady state, preventing continuous configuration changes which might impact performance.
Second, it is important to determine the most effective and efficient reconfiguration action to address a specific problem since choosing the wrong reconfiguration action can make the problem worse. For example, when the performance of the database cluster degrades due to network congestion, adding an extra replica will only cause more network traffic. Which reconfiguration actions are best to decrease the size of the inconsistency window while minimizing the impact on performance is open for research.

6. TENTATIVE RESEARCH PLAN

The tentative research plan can be divided into the following tasks:

1. The first part of the research plan consists of an examination of the parameters that might impact the size of the inconsistency window. These parameters can be: the load on the database, the amount of RAM available, the amount of nodes in the cluster, etc. This research provides a good understanding on how these different parameters influence each other and which parameters can be used during reconfiguration operations to decrease the size of the inconsistency window.

2. In a second step, we need to research different ways to monitor the size of the inconsistency window with minimal impact on the overall performance. Depending on where the database system is deployed, eg., in the public cloud or in the private cloud, different monitoring techniques might be required.

3. Next, we need to examine whether it is possible to derive consistency-related parameters from combining the information in the SLA, the observed consistency measurements and the observed system and network-level performance. For most database systems, the size of the inconsistency window is tweakable using several consistency-related parameters. The appropriate configuration might depend on the used database technology, the workload, size of the database cluster, etc. It is not obvious whether this is possible.

4. Finally, the monitoring support from step two, needs to be combined with tactics to perform autonomous reconfiguration actions. As such continuous monitoring of the inconsistency window can be performed together with autonomous reconfiguration and re-provisioning of the database cluster to operate the database at the minimal cost while satisfying the non-functional requirements defined in the SLA of the application.

7. CONCLUSIONS

Many heterogeneous NoSQL technologies exist. All these system have different architectures and allow tweaking different consistency-related parameters. Since there is a link between consistency guarantees, performance and availability, is it important to choose the right consistency-related parameters to operate the database system at low cost without violating any of the consistency, performance and availability requirements defined in the SLA of the application. As such we try to develop an autonomous system which automatically and continuously configures the consistency-related configuration parameter. As a result, system administrators and application programmers do not have to worry about defining these parameters since they are derived from the SLA.

8. REFERENCES

[1] P. Bailis and A. Ghodsi. Eventual consistency today: limitations, extensions, and beyond. 2013.

[2] J. Baker, C. Bond, J. C. Corbett, J. Furman, A. Khorlin, J. Larson, J.-M. Leon, Y. Li, A. Lloyd, and V. Yushprakh. Megastore: Providing scalable, highly available storage for interactive services. 2011.

[3] D. Bermbach and S. Tai. Eventual consistency: How soon is eventual? an evaluation of amazon s3's consistency behavior. ACM, 2011.

[4] D. Bermbach and S. Tai. Benchmarking eventual consistency: Lessons learned from long-term experimental studies. 2014.

[5] E. Brewer. Cap twelve years later: How the "rules" have changed. *Computer*, 2012.

[6] P. Cassandra. A Netflix Experiment: Eventual Consistency != Hopeful Consistency by Christos Kalantzis. http://planetcassandra.org/blog/a-netflix-experiment-eventual-consistency-hopeful-consistency-by-christos-kalantzis/.

[7] G. F. Coulouris, J. Dollimore, and T. Kindberg. *Distributed systems: concepts and design*. pearson education, 2005.

[8] Datastax. Datastax documentation: Apache Cassandra^{TM} 2.1. http://docs.datastax.com/en/cassandra/2.1/cassandra/gettingStartedCassandraIntro.html, 2015.

[9] S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 2002.

[10] W. Golab, M. R. Rahman, A. AuYoung, K. Keeton, and I. Gupta. Client-centric benchmarking of eventual consistency for cloud storage systems. IEEE, 2014.

[11] W. Golab, M. R. Rahman, A. AuYoung, K. Keeton, and X. S. Li. Eventually consistent: not what you were expecting? 2014.

[12] MongoDB. The MongoDB 3.0 Manual. http://docs.mongodb.org/manual/, 2015.

[13] S. Patil, M. Polte, K. Ren, W. Tantisiriroj, L. Xiao, J. López, G. Gibson, A. Fuchs, and B. Rinaldi. Ycsb++: benchmarking and performance debugging advanced features in scalable table stores. ACM, 2011.

[14] W. Vogels. Eventually consistent. 2009.

[15] H. Wada, A. Fekete, L. Zhao, K. Lee, and A. Liu. Data consistency properties and the trade-offs in commercial cloud storage: the consumers' perspective. 2011.